

Rec'd PGT/PTO

25 APR 2005

PCT/DE 03/03524

BUNDESRREPUBLIK DEUTSCHLAND

10/532613

REC'D 09 DEC 2003

WIPO

PCT

PRIORITY DOCUMENT  
SUBMITTED OR TRANSMITTED IN  
COMPLIANCE WITH  
RULE 17.1(a) OR (b)



**Prioritätsbescheinigung über die Einreichung  
einer Patentanmeldung**

**Aktenzeichen:**

102 49 676.5

**Anmeldetag:**

24. Oktober 2002

**Anmelder/Inhaber:**

Siemens Aktiengesellschaft, München/DE

**Bezeichnung:**

Einrichtung der programmierbaren Logik

**IPC:**

G 06 F 9/06

VERIFIED COPY OF  
PRIORITY DOCUMENT

Die angehefteten Stücke sind eine richtige und genaue Wiedergabe der ursprünglichen Unterlagen dieser Patentanmeldung.

München, den 5. November 2003  
Deutsches Patent- und Markenamt  
Der Präsident  
Im Auftrag

Kohle

BEST AVAILABLE COPY

## Beschreibung

## Einrichtung der programmierbaren Logik

5 Die Erfindung bezieht sich auf eine Einrichtung der programmierbaren Logik mit mehreren Logikblöcken mit konfigurierbaren Eigenschaften, die jeweils mindestens eine logische Verarbeitungseinheit mit Funktionsprogrammen und Schnittstellen zu den jeweils anderen Logikblöcken umfassen, mit mindestens  
10 einer den Logikblöcken zugeordneten Eingabe- und Ausgabeeinheit sowie mit Mitteln zum Verknüpfen der Logikblöcke untereinander und mit der Verarbeitungseinheit und der Ein-/Ausgabeeinheit. Eine derartige Logikeinrichtung ist der  
US 4,870,302 A zu entnehmen.

15

Programmierbare logische Bausteine von herkömmlichen Logikeinrichtungen wie insbesondere Prozessoren führen Programme aus, die aus einem Speicher geladen werden. Die auszuführende Software in Form von Befehlswörtern ist dort als Maschinenbefehl  
20 abgelegt. Diese Befehlswörter werden geladen, analysiert, interpretiert und in einer Verarbeitungseinheit ausgeführt. Dabei löst die Verarbeitung eines einzigen Befehlswortes eine Vielzahl von Einzelaktionen in der Logikeinrichtung aus.

Die Basisstruktur und -organisation bekannter digitaler Logikeinrichtungen, insbesondere von Computern mit Mikroprozessoren, beruht auf der Konzeption des sogenannten „Von-Neumann-Rechners“. Dessen Zentraleinheit CPU („Central Processing Unit“), d.h. dessen Computerkern, umfasst in ihrer  
30 Minimalkonfiguration als Hauptbestandteile einen Hauptspeicher, eine Steuereinheit und eine Verarbeitungseinheit (bzw. Rechenwerk):

- Der Hauptspeicher speichert Befehlswörter (Programmdaten)  
35 und Verarbeitungsdaten (Operantenwörter) und stellt diese auf Aufforderung zur Verfügung. Ferner nimmt der Hauptspeicher Zwischen- und Endresultate der Verarbeitung auf.

Hauptspeicher können durch flüchtige oder nicht-flüchtige Speicher realisiert werden.

- Die Steuereinheit organisiert die Reihenfolge, in der Befehlswörter abgearbeitet werden. Sie fordert Befehlswörter aus dem Hauptspeicher an und veranlasst deren Ausführung in der Verarbeitungseinheit. Außerdem analysiert sie die Befehlswörter und veranlasst die Lieferung von Verarbeitungsdaten an die Verarbeitungseinheit.
- Die Verarbeitungseinheit führt die Operation an den Verarbeitungsdaten aus und liefert entsprechende Resultatwörter an den Hauptspeicher. Für jede Operation enthält die Verarbeitungseinheit ein Mikroprogramm, das die benötigten Übertragungswege freischaltet. Die Verarbeitungseinheit wird durch die Steuereinheit auf die jeweilige Operationsart, d.h. auf den abzuarbeitenden Befehl, eingestellt. Der Zentraleinheit sind Peripheriegeräte zugeordnet, bei denen es sich um externe Speicher sowie um Ein- und Ausgabegeräte handeln kann. Die angegebenen Hauptkomponenten der Zentraleinheit können physikalisch getrennt sein; zu- meist sind sie jedoch auf einem gemeinsamen Prozessorchip mit einem Cache oder beispielsweise einem embedded ROM realisiert.

Eine genauere Betrachtung der Programmierung einer solchen programmierbaren Logikeinrichtung PLD („Programmable Logic Device“) mit den erwähnten Grundkomponenten des Von-Neumann Rechners zeigt, dass das Programm sowie die Initialisierungsdaten im preiswerten Speicher (mit minimal 1 Transistor pro Speicherzelle) untergebracht sind und erst dann in die CPU zur Ausführung gelangen, wenn sie an der Reihe sind. Die preiswerte Speicherung einerseits und die „Wiederverwendung“ der aufwendigeren CPU-Schaltkreise wie ALU („Arithmetic Logic Unit“) für alle möglichen Instruktionen andererseits sind die positiven Beiträge zu einer sogenannten „funktionalen Dichte“ der Prozessor-basierten Rechner. Hierbei ergibt die zeitliche Sequenz der Bearbeitung natürlich einen negativen Beitrag. Die funktionale Dichte kann dabei als mittlere Anzahl von ak-

tiven Gatteräquivalenten pro Siliziumfläche und Zeit definiert werden.

Bei bekannten programmierbaren Logikeinrichtungen (PLDs) mit  
5 Rechenkapazität, Speicher und Ein- und Ausgabeeinheit I/O („Input/Output“), wie sie z.B. aus der eingangs genannten USA-Schrift zu entnehmen ist, wird deren Struktur einmal programmiert, und der Programm-(wie Daten-)Inhalt wird an der Struktur selbst gespeichert. Diese Form der Speicherung ist  
10 aufwendig, denn um den Speicher selbst müssen zwecks schneller Umsetzung in geschaltete Datenpfade eine Menge von zusätzlichen Transistoren hinzukommen. Es existieren zwar nur wenige Angaben über die Ausnutzung des Siliziums; man sollte von Faktoren 20 bis 40 im Verhältnis Gesamtzahl/sichtbarer  
15 Transistorkapazität (im Sinne der Schaltfunktion) ausgehen.

Da die Programmierbarkeit in Form eines Speichers mit Umsetzung in geschaltete Datenpfade recht teuer ist, wie im Fall des Rechners aber bislang das komplette Programm im Baustein  
20 lagert, erreichen die PLDs trotz Potenzials für hohe funktionale Dichte „nur“ den Faktor 10 mehr als Prozessoren. Das bedeutet, dass der Geschwindigkeitsgewinn in PLDs gegenüber Prozessorlösungen teuer erkauft ist.

Aufgabe der vorliegenden Erfindung ist es deshalb, die programmierbare Logikeinrichtung mit den eingangs genannten  
30 Merkmalen dahingehend auszugestalten, dass bei hoher funktionaler Dichte eine hohe Geschwindigkeit des PLDs mit einfachen Mitteln zu erreichen ist.

Diese Aufgabe wird erfindungsgemäß mit den in Anspruch 1 angegebenen Maßnahmen gelöst. Diese Maßnahmen umfassen eine Rekonfigurierbarkeit der Logikblöcke während des Betriebs der Logikeinrichtung dadurch, dass die Verknüpfungsmittel zusätzlich  
35 wenigstens einen konfigurierbaren Umschalt-Logikblock aufweisen, mit dem die Verknüpfung wenigstens einiger der rekonfigurierbaren Logikblöcke untereinander und/oder mit der

Verarbeitungseinheit und/oder der Ein-/Ausgabeeinheit erfolgt.

Die mit einer solchen Architektur verbundenen Vorteile sind insbesondere darin zu sehen, dass bei begrenzter Anzahl von nunmehr rekonfigurierbaren Logikblöcken die Verbindungen zwischen diesen Blöcken und/oder zu externen Bausteinen wie Speichereinheiten oder Mikroprozessoren wie z.B. zu der Verarbeitungseinheit und/oder zu der Ein-/Ausgabeeinheit nicht ein für alle Male fest vorgegeben sind, sondern dass diese Verbindungen mit Hilfe des zusätzlich vorgesehenen Umschalt-Logikblocks während des Betriebs bedarfsmäßig erstellt werden können. Damit wird es ermöglicht, dass von einzelnen Blöcken unterschiedliche Funktionen zu unterschiedlichen Zeitpunkten auszuführen sind; d.h., deren Ausnutzung wird entsprechend erhöht. Damit verbunden ist eine entsprechende Performance-Verbesserung der gesamten Logikeinrichtung, verglichen mit einem Mikroprozessor, bzw. kein Performance-Verlust im Vergleich zu herkömmlichen PLDs.

Vorteilhafte Ausgestaltungen der erfindungsgemäßen Logikeinrichtung gehen aus den abhängigen Ansprüchen hervor.

So kann insbesondere die Verknüpfung wenigstens einiger der rekonfigurierbaren Logikblöcke einem vorgegebenen Kontext entsprechend erfolgen.

Ferner kann vorteilhaft der Umschalt-Logikblock wenigstens einen Zustandsspeicher aufweisen, der Informationen bezüglich der Funktionen der einzelnen rekonfigurierbaren Logikblöcke enthält, so dass die Verknüpfung der ausgewählten rekonfigurierbaren Logikblöcke gemäß den Funktionsinformationen des aktuellen Zustands erfolgt.

Die den erfindungsgemäßen Maßnahmen zu Grunde liegenden Überlegungen werden nachfolgend unter Bezugnahme auf die Zeichnung noch weiter erläutert. Dabei zeigen

deren Figur 1 die Grundstrukturen von vier einfachen, bekannten Automatentypen,

deren Figur 2 die Grundstruktur einer sogenannten „Sequential Finite State Machine“,

5 deren Figur 3 den schematischen Aufbau einer partiell rekonfigurierbaren PLD

und

deren Figur 4 den schematischen Aufbau einer erfindungsgemäßen Logikeinrichtung.

10

Im Folgenden werden aus den Maschinen vom sogenannten „*Finite State*“-Typ die sogenannten „*Sequential Finite State Machines*“ (SFSM) abgeleitet. Das hierfür aufgezeigte Modell ist dazu geeignet, eine Sequenz von Konfigurationen im PLD zu definieren, und genau diese Sequenz kann ohne Performance-Verlust  
15 eine wesentlich höhere funktionale Dichte (gleichbedeutend mit drastisch gesenkten Kosten für Herstellung und Betrieb) eines PLDs erzeugen.

## 20 Abschnitt I („Sequential Finite State Machines“)

Eines der „klassischen“ Denkmodelle für eine Hardwareentwicklung besteht in den einfachen endlichen Automaten, im Folgenden „*Finite State Machines*“ (FSM) genannt.

Dieses eng mit theoretischen Konzepten (insbesondere des sogenannten „Deterministischen endlichen Automaten“) verwandte Modell ist exakt wie folgt definiert:

### 30 Definition 1:

Eine FSM besteht aus einem 6-Tupel  $\{A, X, Y, f, g, a_0\}$ .  $A = \{a_0, a_1, \dots, a_M\}$  ist hierbei die endliche Menge der Zustände, wobei  $a_0$  den Startzustand bedeutet.

35  $X = \{X_1, \dots, X_K\}$  ist die endliche Menge der Eingangsvektoren mit  $X_I = (x_1, \dots, x_L)_1$ , wobei  $x_1 \in \{0, 1, -\}$ .

$Y = \{Y_1, \dots, Y_N\}$  ist die endliche Menge der Ausgangsvektoren mit  $Y_j = (y_1, \dots, y_N)_j$ , wobei  $y_h \in \{0, 1, -\}$ .

$f: A \times X \rightarrow A$  heißt Transitionsfunktion (Next State Decoder),

5  $g: A \times X \rightarrow Y$  heißt Ausgangsfunktion (Output Decoder),  
 $t = \text{Zeiteinheit}$ .

Die in dieser Definition genannten Funktionen werden durch Schaltnetze realisiert, die den algorithmischen Zusammenhang  
10 zwischen den (im Wesentlichen binären) Eingangs- und Zustandsvektoren darstellen. Hierzu ist allerdings notwendig, dass auch die Zustände binär codiert werden, was in Definition 1 noch nicht der Fall war.

15 Für die Zustandsautomaten werden weiterhin drei wesentliche Untertypen (siehe Figur 1) unterschieden, deren Einfluss auf die Komplexität der Funktionen in der Praxis gegeben ist. Ein Entwickler wird folgende Schritte durchführen:

- 20 1. *Festlegen der Signale*: Das Schaltwerk wird als „Black Box“ mit den erforderlichen Eingangs- und Ausgangssignalen skizziert.
2. *Entwerfen des Zustandsdiagramms*: Dieser Schritt ist der eigentliche Kern der Synthese, da hier das zu lösende Problem formal beschrieben wird. Als Zustandsdiagramm sind Zustandsgraphen, Programmablaufpläne oder auch Schaltwerkstabellen möglich.
3. *Aufstellen der Schaltwerkstabelle* als formaler Ausgangspunkt für alle weiteren Operationen.
- 30 4. *Zustandsminimierung*: Die Minimierung der Anzahl der Zustände soll eine Vereinfachung des Designs erreichen.
5. *Zustandscodierung*: Bei synchronen Schaltwerken mit synchronisierten Eingängen (nicht für Mealy-Automat gemäß Figur 1d) lassen sich beliebige Codierungen für die Zustände aus  $Z$  angeben. Beim Medwedjew-Automat gemäß Figur 1b müssen die Codierungen allerdings mit den gewünschten Ausgangssignalen übereinstimmen; beim Moore-Automat gemäß
- 35

Figur 1c hingegen können die Codierungen so gewählt werden, dass sich vereinfachte Schaltnetze für Next State Decoder (f) und Output Decoder (g) ergeben.

#### 6. Berechnung von Folgezustands- und Ausgangsschaltnetzen:

5 Zur konkreten Berechnung des Folgezustandsschaltnetzes muss ein Register- bzw. Flipfloptyp gewählt werden, da deren Eingänge (T, D, RS oder JK) durch dieses Schaltnetz belegt werden, aber unterschiedliche Funktionalitäten zeigen.

#### 10 7. Realisierung des Schaltwerks und Test

Aus der Praxis weiß man nun, dass die Implementierung als einfache Maschine - man könnte sie auch als „flache Maschine“ bezeichnen - nicht unbedingt die beste ist. Ein kooperieren-  
 15 der Automat, bestehend aus mehreren einfachen Automaten, die miteinander gekoppelt sind, kann dies häufig wesentlich besser im Sinne von flächeneffizienter. Zudem sollte bedacht werden, dass der vorstehend skizzierte Designfluss nicht unbedingt in der angegebenen Weise durchgeführt werden muss.  
 20 Gerade mit zunehmender Beschreibung in Hochsprachen wie VHDL ist eine Hinwendung zu mehr algorithmischem Stil zu erkennen.

Unabhängig davon soll jedoch die FSM als das grundlegende Modell angenommen werden. Man kann sich nun vorstellen, nicht nur eine FSM zu haben, sondern mehrere, von denen exakt eine zu einem Zeitpunkt aktiv ist. Ein ausgezeichneter Teil (der sogenannte „Sequencer“) schaltet dann in Abhängigkeit von Eingangssignalen oder erreichten Resultaten zwischen den einzelnen FSMs um.

30

Ausgehend von dieser FSM wird nachfolgend die Sequential Finite State Machine (SFSM) korrekt definiert und dargestellt :

#### Definition 2:

35

Eine Sequential Finite State Machine (SFSM) besteht aus einem 5-Tupel  $(B, B_0, C, V, h)$ . Hierbei stellt  $B = \{B_0,$

...  $B_K$  eine endliche Menge von Finite State Machines (FSM) dar,  $B_0$  ist die Start-FSM.  $C = \{C_0, \dots, C_K\}$  beschreibt eine endliche Menge von Zuständen zur Kenntlichmachung der aktuellen FSM.  $V = \{V_1, \dots, V_N\}$  ist die endliche Menge der (zusätzlichen) Eingangsvektoren mit  $V_I = (v_1, \dots, v_L)_I$ , wobei  $v_i \in \{0, 1, -\}$ .  $h: B \times V \rightarrow B$  heißt FSM-Transitionsfunktion (Next FSM Decoder).

Der wesentliche Vorteil dieser SFSM (vgl. hierzu auch Figur 2) liegt erst einmal in der Modellierung. Ein Designer hat die Chance, sein Design in kleinere Portionen zu teilen. In der technischen Ausführung wird man dann versuchen, die (weiterhin endliche) Menge der Zustände aus allen FSMs  $B_K$  auf eine einheitliche Zustandscodierung abzubilden. Zusätzlich hierzu muss eine Codierung für die  $C_K$  der aktuellen FSM  $B_K$  mitgeführt werden, um für die Funktion  $h$  die Berechnung der nächsten FSM zu ermöglichen.

Andererseits zeigt ein Blick auf Figur 2, dass eigentlich nur wenig gewonnen ist. Falls der Takt für alle Register identisch ist, dann wurden in Figur 1a die Register und das Schaltnetz  $f(u^*, x^*)$  (Next State Decoder), \* bedeutet Gesamtmenge) in Teilmengen von Registern mit  $f(u, x, c)$  und  $h(u, v, c)$  eingeteilt, die in der Realisierung wieder zusammengeführt werden. Minimaler wird die Lösung dadurch kaum, denn in einem PLD heutiger Bauart müssen alle Teile, also alle Teil-FSM, auf dem Baustein integriert werden. Hier setzen nun die erfindungsgemäßen Überlegungen an.

Es existieren zwar wenige FPGAs („Field-Programmable Gate Arrays“; vgl. z.B. „Spektrum der Wissenschaft“, August 1997, Seiten 44 bis 49), die dynamisch rekonfigurierbar sind, jedoch nur partiell. Diese rekonfigurierbare Eigenschaft könnte man ausnutzen, indem in einem permanenten Teil  $h(u, v, c)$  und die Register zur FSM-Codierung sowie im nachladbaren Teil die aktuelle FSM geladen und ausgeführt werden. Das Problem hier-

bei ist der Performanceverlust beim Nachladen. Dies ist aus dem nachfolgenden Beispiel ersichtlich:

### Beispiel

5

Die gern genommene Straßenverkehrsampel als Beispiel für eine Finite State Machine kann auch sehr gut als Beispiel für eine Sequential FSM genommen werden, wenn man an einen Nachtmodus denkt. Das Wort Modus zeigt schon den Weg: Im Umschalten zwischen einzelnen FSMs sollte so etwas wie ein Moduswechsel liegen, und Tag- und Nachtmodus schließen sich nun einmal gegenseitig aus.

10

15

FSM1 integriert nun die Tagampel, FSM2 die Nachtampel (z.B. gelbes Blinken für die Nebenstraßen), und im übergeordneten Sequencer wird an Hand eines Zeitsignals entschieden, welche FSM zur Ausführung kommt und welche ruht. Der Sequencer wird als FSM0 implementiert.

20

Die Beschreibung für hierfür erforderliche drei FSM (0 .. 2) wird getrennt durchgeführt, dann aber auf einem PLD integriert, so dass im Bereich 0 die FSM0, 1 FSM1 und 2 FSM2 liegen würde. Ein Flächengewinn wäre damit nicht zu verzeichnen.

30

Im nachladbaren Fall müsste FSM0 permanent zur Verfügung stehen, außerdem ein Bereich, der das Maximum von {FSM1, FSM2} aufnehmen könnte. In diesen Bereich wäre dann ständig eine von zweien geladen, und das Umschalten würde eine Rekonfiguration nach sich ziehen.

35

Zur Verdeutlichung des Vorteils bei dynamisch rekonfigurierbaren PLDs, wie sie Logikeinrichtungen nach der Erfindung darstellen, wird die Übergangsfunktion  $f(u, x, c)$  (vgl. Figur 2) nunmehr als  $f_c(u, x)$  (Äquivalentes gilt für Funktion  $g$ ) bezeichnet. Mit dieser Darstellung soll die Unterschiedlichkeit in der Abhängigkeit demonstriert werden:  $u$  und  $x$  bewir-

ken eine ständige, mit dem Takt verbundene Fortentwicklung des Zustands der Maschine, während c die Abhängigkeit vom wesentlich selteneren Moduswechsel beschreibt.

5 Die Rekonfiguration selbst würde vergleichsweise viel Zeit in Anspruch nehmen; es wurde nun erkannt, dass eine wesentlich günstigere Lösung in einer erweiterten Architektur von PLDs besteht, in denen zugleich mehrere Programme dynamisch umschaltbar gespeichert sind. Eine entsprechende Logikeinrichtung  
10 tung könnte als Multi-Plane-PLD bezeichnet werden; um jedoch dem Aspekt des jeweiligen „Kontext“ Rechnung zu tragen und darzustellen, dass auch nur Teilbereiche von Umschaltvorgängen betroffen sind, wird hier der Name „Dynamical Multi-Context PLDs (dMC-PLDs)“ gewählt.

15

## **Abschnitt II ( Erfindungsgemäße Architektur von „Dynamical Multi-Context PLDs“)**

Zunächst sei ein Problem angesprochen, dass sowohl für klassische dynamisch rekonfigurierbare als auch für dMC-PLDs  
20 gilt. Wird nämlich lediglich ein Teilbereich umgeschaltet, müssen dessen Schnittstellen nach außen hin, also insbesondere in Richtung der verbleibenden Teile, konstant bleiben. Zugleich sind einige Funktionsprogramme (sogenannte „Routingressourcen“) von dem Umschalten mit betroffen, da innerhalb des umzuschaltenden Blocks nicht nur Logik, sondern auch Verbindungen liegen müssen.

Konsequenterweise wird im Folgenden für die erfindungsgemäße  
30 Logikeinrichtung ein PLD-Block als eine rekonfigurierbare Einheit, bestehend aus logischen Verarbeitungseinheiten mit Routingressourcen und Schnittstellen (sogenanntes „Routing“) zu anderen Blöcken definiert; diese Definition stimmt mit den in der Praxis üblichen, partiell rekonfigurierbaren Bausteinen  
35 überein. Die Definition dieses Blocks weicht dabei von der üblichen Definition der logischen Blöcke ab.

Wenn man sich jetzt einmal einen fiktiven PLD mit solchen Blöcken anschaut, erhält man eine Architektur, wie sie aus Figur 3 hervorgeht. In der Figur sind bezeichnet mit 2 eine partiell rekonfigurierbare PLD-Einrichtung bzw. -Struktur, mit 3A bis 3D vier PLD-Blöcke (= Logikblöcke), mit 4 ein (globales) Routing, das eine Verarbeitungseinheit mit Funktionsprogrammen und Schnittstellen zu den Blöcken bildet, sowie mit 5 ein I/O-Bereich, der Mittel zum Verknüpfen der Logikblöcke mit der Verarbeitungseinheit und einer Ein-/und Ausgabeeinheit darstellt. Man kann sich diese Architektur so vorstellen, dass - einmal abgesehen von den I/O-Ressourcen 5 am Außenrand - jeder PLD-Block 3A bis 3D einem vollwertigen CPLD („Complex PLD“) oder FPGA entspricht. Derartige Architekturen existieren bereits, z.B. „Cypress Delta39k“, „Lattice-Vantis Godfather“-Architektur; sie sind jedoch nicht im Betriebszustand der PLD-Einrichtung rekonfigurierbar.

Bei der in Figur 4 schematisch angedeuteten dMC-PLD-Architektur einer Logikeinrichtung 7 nach der Erfindung wird die PLD-Struktur 2 nach Figur 3 durch einen zusätzlichen PLD überlagert. Dieser als Umschalt-Logikblock 8 bezeichnete Extra-PLD ist bei bekannten Logikeinrichtungen nicht vorhanden und soll die Umschaltfunktion  $h(u, v, c)$  (vgl. Figur 2) übernehmen. Er stellt folglich einen Teil von Verknüpfungsmitteln dar, mit denen die Verknüpfung wenigstens einiger der Logikblöcke 3A bis 3D untereinander und/oder mit der Verarbeitungseinheit 4 und/oder der Ein-/Ausgabeeinheit 5 erfolgt. Dies bedeutet, dass die Ausgänge in Figur 4 einem gespeicherten Kontext  $c$  entsprechen und die Auswahl der aktuellen Funktionen  $f_c(u, x)$  und  $g_c(u, x)$  steuern.

Für diesen Extra-PLD ergibt sich damit eine Struktur, die aus vielen Eingängen  $(u, x)$  wenige, ausschließlich gespeicherte Ausgänge berechnet.

Die sich aus diesem Aufbau ergebenden Vorteile sind darin zu sehen, dass die funktionale Dichte und damit die Ausnutzbar-

keit gesteigert werden bzw. die Kosten pro Anwendung zu senken sind. Folgende grobe Abschätzung kann dafür gegeben werden:

- 5 Es sei angenommen, dass die Ausnutzung der verwendeten Transistoren 1:20 ist; d.h., von 20 eingesetzten Transistoren ist tatsächlich nur einer in der (programmierbaren) logischen Funktion sichtbar. Für den Extra-PLD 8 lassen sich 20 % Overhead veranschlagen, da es sich um einen Logikblock handelt, der nicht zur eigentlichen Funktionalität beiträgt, sondern nur die anderen Blöcke 3A bis 3D umschaltet. Für die Speicherung und Decodierung der zusätzlichen Programme (es werden lediglich 4 gespeicherte Funktionen pro PLD-Block angenommen) werden weitere 6 Transistoren berechnet (4 Transistoren zur Speicherung und 2 zur Auswahl 1 aus 4). Daraus ergibt sich eine neue Ausnutzung von 4:30, mithin eine Steigerung der Dichte gegenüber bekannten Ausführungsformen um 166 %.
- 10
- 15
- 20 Gegebenenfalls könnte sich eine Schwierigkeit dadurch ergeben, dass sich auf Grund mehrerer Konfigurationen, aus denen eine aktuelle auszuwählen ist, der kritische Pfad innerhalb eines PLD-Blocks um einen Transistor verlängern kann, was mit der Erniedrigung der maximalen Taktfrequenz einhergehen würde. Hierzu muss zwischen sogenannter SRAM-basierter Speicherung einerseits und sogenannter FLASH-EEPROM bzw. Anti-Fuse andererseits unterschieden werden.

- 30 Für SRAM-basierte Verfahren wird mit der US 6,011,740 A ein Verfahren vorgeschlagen, die Speicherung mehrerer Konfigurationen in einem Schatten-RAM z.B. als Ringspeicher zu realisieren und die aktuell benötigte Information in eine Arbeitszelle zu laden. Dieses Verfahren garantiert eine maximale Arbeitsgeschwindigkeit.

35

Für Flash- und Antifuse-Speicherung könnte als eigentliche Zelle ebenfalls eine nachladbare SRAM-Zelle in Betracht kom-

men, so dass die obengenannte Lösung zum Tragen kommt; dies bedeutet jedoch einen zusätzlichen Aufwand und möglicherweise eine Verringerung der Geschwindigkeit, bedingt durch den Technologiewechsel bei der Speicherung. Bei diesen Technologien wird tatsächlich ein weiterer Path-Transistor, der den aktuellen Kontext decodiert, im Pfad eingeführt, gegebenenfalls in Form von Dual-Gate-FETs.

Die genaue Einteilung, welche Blockgröße und Extra-PLD-Größe für die Vielzahl an Anwendungen von Vorteil sind, kann durch Simulationen an Beispielen ermittelt werden.

Zu einer Realisierung einer erfindungsgemäßen Logikeinrichtung nach Figur 4 kann man sich vorstellen, dass die beiden schematisch dargestellten Schichten, nämlich die mit dem Extra-PLD 8 (= Umschalt-Logikblock) und die mit den Logikblöcken 3A bis 3D, jeweils in einer Ebene E2 bzw. E1 zumindest weitgehend äquivalent sind und sich nicht nur gegenseitig umschalten, sondern vielmehr auch die Programmierung des jeweils anderen Teils erzeugen. Umschaltung bedeutet in diesem Fall, dass zwischen vorbestimmten Programmen ausgewählt wird. Gegenseitige Programmierung hingegen kann zur Anpassung (Evolution) an die jeweilige Umgebung genutzt werden. Ein noch sehr einfaches Beispiel kann eine digitale PLL sein, die eine Frequenz mittels Laufzeiteffekten an einen Mastertakt anpasst.

## Patentansprüche

## 1. Einrichtung der programmierbaren Logik

- 5 - mit mehreren Logikblöcken mit konfigurierbaren Eigenschaften, die jeweils mindestens eine Verarbeitungseinheit mit Funktionsprogrammen und Schnittstellen zu den jeweils anderen Logikblöcken umfassen,  
- mit mindestens einer den Logikblöcken zugeordneten Ein-  
10 /Ausgabeeinheit,  
und  
- mit Mitteln zum Verknüpfen der Logikblöcke untereinander und mit der Verarbeitungseinheit und der Ein-/Ausgabeeinheit,  
15 g e k e n n z e i c h n e t durch eine Rekonfigurierbarkeit der Logikblöcke (3A bis 3D) während des Betriebs der Logikeinrichtung (7) dadurch, dass die Verknüpfungsmittel zusätzlich wenigstens einen konfigurierbaren Umschalt-Logikblock (8) aufweisen, mit dem die Verknüpfung wenigstens einiger der  
20 rekonfigurierbaren Logikblöcke (3A bis 3D) untereinander und/oder mit der Verarbeitungseinheit(4) und/oder der Ein-/Ausgabeeinheit (5) erfolgt.

2. Einrichtung nach Anspruch 1, d a d u r c h g e -  
k e n n z e i c h n e t , dass die Verknüpfung wenigstens einiger der rekonfigurierbaren Logikblöcke (3A bis 3D) einem vorgegebenen Kontext (c) entsprechend erfolgt.

3. Einrichtung nach Anspruch 1 oder 2, d a d u r c h  
30 g e k e n n z e i c h n e t , dass der Umschalt-Logikblock (8) wenigstens einen Zustandsspeicher aufweist, der Informationen bezüglich der Funktionen der einzelnen rekonfigurierbaren Logikblöcke (3A bis 3D) enthält, und dass die Verknüpfung der ausgewählten rekonfigurierbaren Logikblöcke gemäß  
35 den Funktionsinformationen des ausgewählten Zustands erfolgt.

4. Einrichtung nach einem der vorangehenden Ansprüche,  
g e k e n n z e i c h n e t durch eine Ebene (E1) mit den  
rekonfigurierbaren Logikblöcken (3A bis 3D) und eine Ebene  
(E2) mit dem Umschalt-Logikblock (8), wobei die Ebenen zumin-  
5 dest weitgehend äquivalent aufgebaut sind.

## Zusammenfassung

## Einrichtung der programmierbaren Logik

- 5 Die Einrichtung (7) der programmierbaren Logik umfasst mehrere Logikblöcke (3A bis 3D) mit konfigurierbaren Eigenschaften und Mittel zum Verknüpfen der Logikblöcke untereinander und mit einer Verarbeitungseinheit (4) und einer Ein-/Ausgabeeinheit (5). Eine Rekonfigurierbarkeit der Logikblöcke (3A
- 10 bis 3D) während des Betriebs der Logikeinrichtung (7) ist dadurch gegeben, dass die Verknüpfungsmittel zusätzlich wenigstens einen konfigurierbaren Umschalt-Logikblock (8) aufweisen, mit dem die Verknüpfung wenigstens einiger der rekonfigurierbaren Logikblöcke (3A bis 3D) untereinander und/oder
- 15 mit der Verarbeitungseinheit (4) und/oder der Ein-/Ausgabeeinheit (5) erfolgt.

FIG 4

FIG 1a

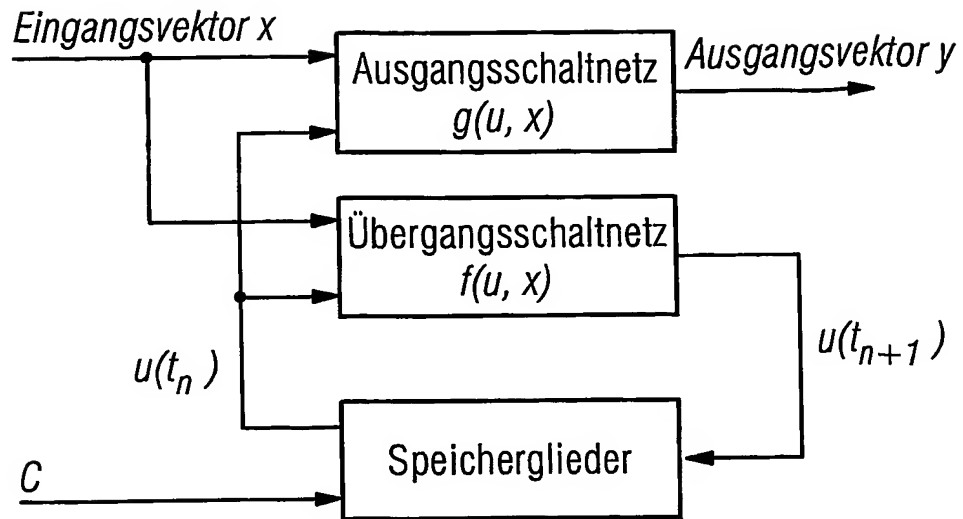


FIG 1b

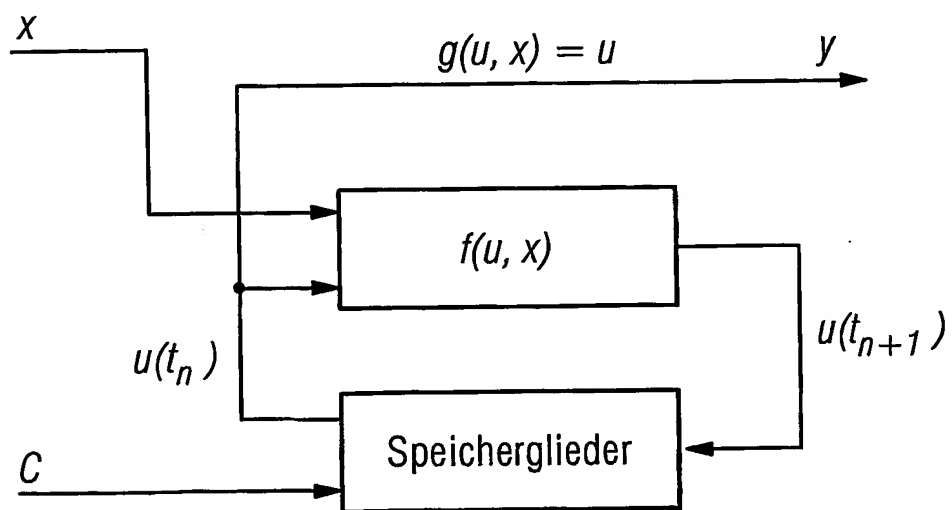


FIG 1c

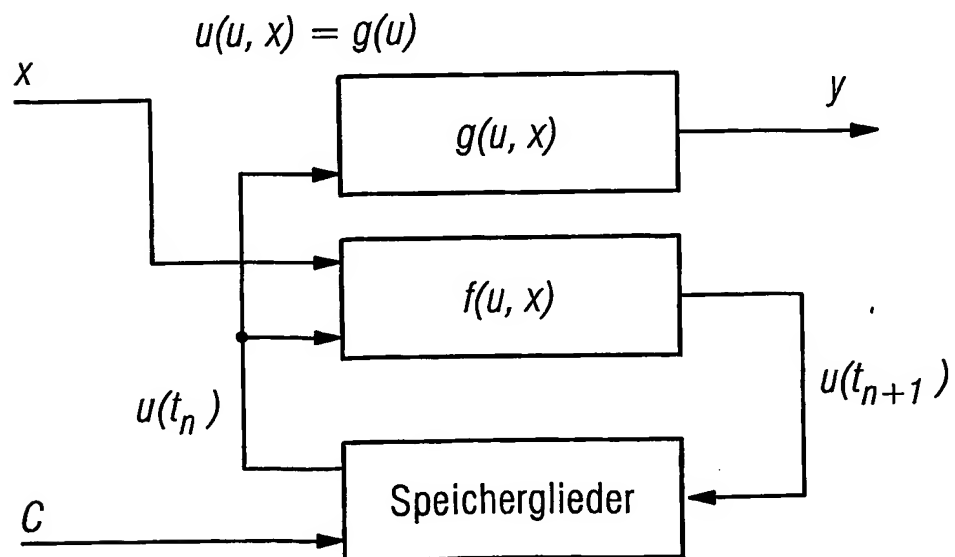


FIG 1d

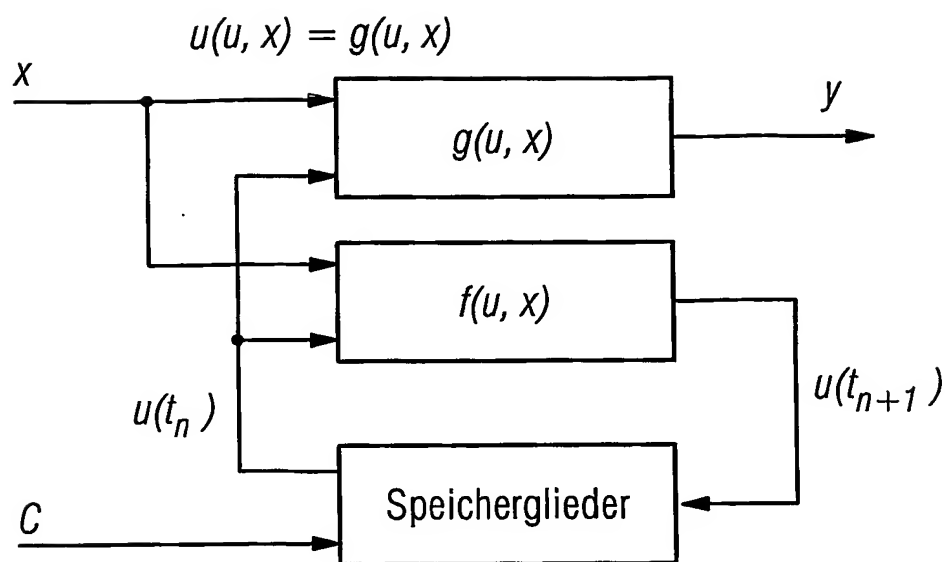


FIG 2

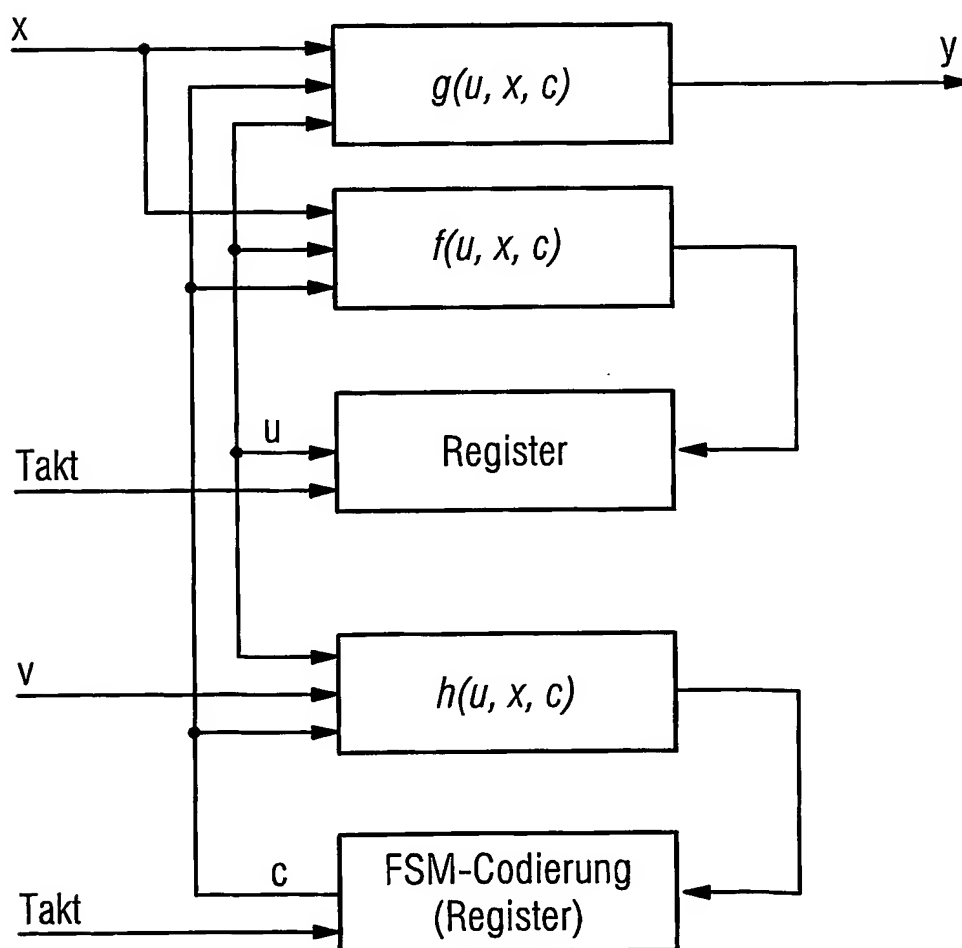


FIG 3

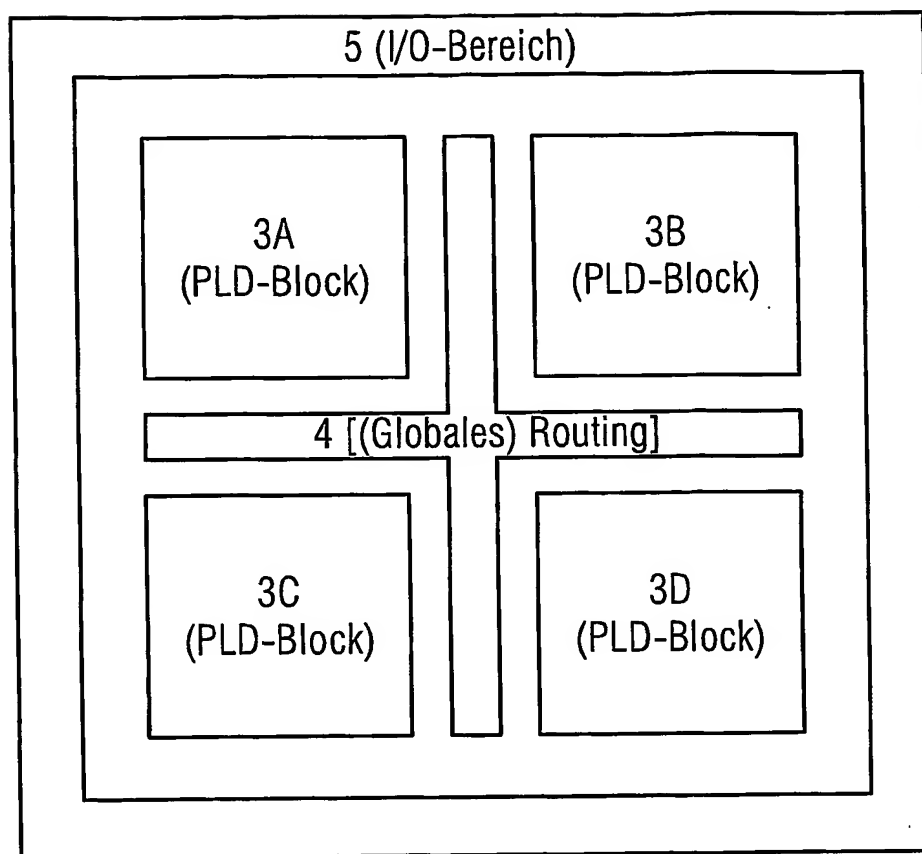
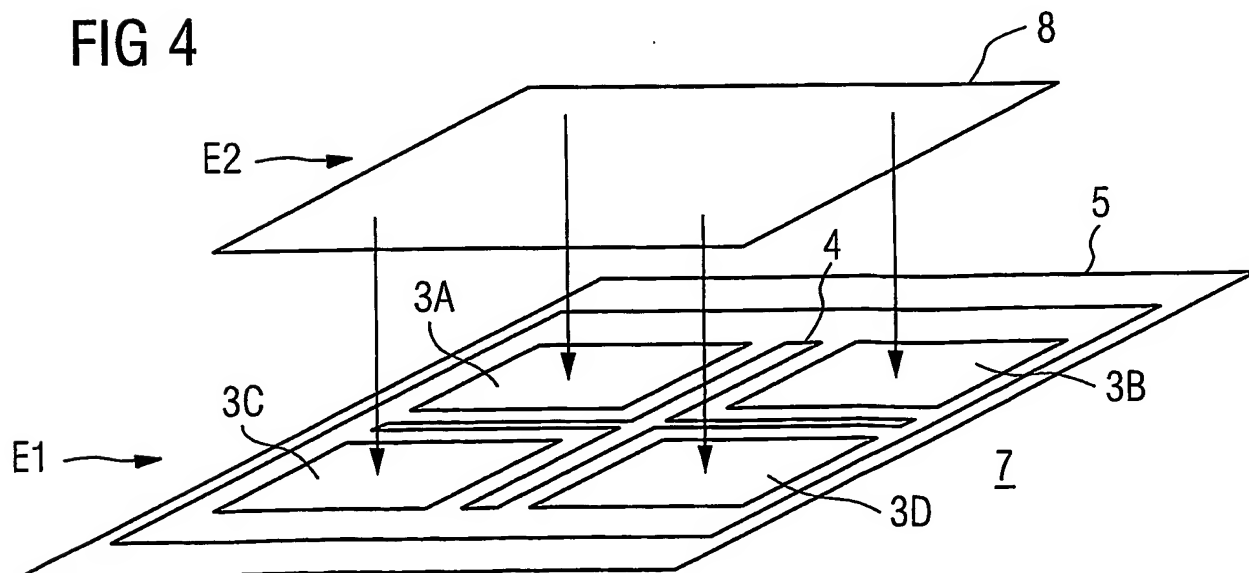


FIG 4



**This Page is Inserted by IFW Indexing and Scanning  
Operations and is not part of the Official Record**

**BEST AVAILABLE IMAGES**

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

- ☒ **BLACK BORDERS**
- ☐ **IMAGE CUT OFF AT TOP, BOTTOM OR SIDES**
- ☐ **FADED TEXT OR DRAWING**
- ☐ **BLURRED OR ILLEGIBLE TEXT OR DRAWING**
- ☐ **SKEWED/SLANTED IMAGES**
- ☐ **COLOR OR BLACK AND WHITE PHOTOGRAPHS**
- ☐ **GRAY SCALE DOCUMENTS**
- ☐ **LINES OR MARKS ON ORIGINAL DOCUMENT**
- ☐ **REFERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY**
- ☐ **OTHER:** \_\_\_\_\_

**IMAGES ARE BEST AVAILABLE COPY.**

**As rescanning these documents will not correct the image problems checked, please do not report these problems to the IFW Image Problem Mailbox.**